Smartrouter Scoping Project — Full Report

2026-02-19

Smartrouter Scoping Project Plan

Last Updated: 2026-02-19 **Status:** In Progress

Overview

Field	Details
Project Name	Smartrouter Scoping Project
Start Date	2026-02-18
Target Completion	TBD
Owner	TBD
Status	Planning

Goals & Success Criteria

Purpose

This is a **scoping and assessment project**. The goal of this phase is to nail down everything that needs to be done and produce a clear, detailed estimate of the effort required to integrate Athia AI/ML into Deuna's payments service. No implementation is happening yet.

Phase 0 Deliverables (Current Focus)

- Full understanding of Deuna's data, schema, and existing routing infrastructure
- Detailed breakdown of all work required across P-01 through P-05 use cases
- Effort estimates per workstream (engineering, data, infra, ML)
- Risk identification and open questions resolved
- Clear recommendation on what to build and in what order

Success Criteria (Phase 0)

- All open questions answered
- Effort estimate produced with confidence
- All access and dependencies identified and documented
- Stakeholder alignment on scope, timeline, and approach

Longer-Term Success Criteria (Post-Scoping)

- Measurable approval lift
- Stability during PSP outages
- Latency target: p95 < 200ms

In Scope (Phase 0 — Assessment Only)

- Understand Deuna's data, schema, and existing routing rules
- Assess Athia platform gaps vs. what's needed
- Size effort for: Processor/Message selector, Smart Retry logic, Feedback Loop
- Identify all dependencies, blockers, and risks

Out of Scope (Phase 0)

- Any implementation or code delivery
- 3DS optimization (Phase 2)
- User-facing messaging (Phase 3)
- Installment optimization

Stakeholders

See TEAMS.md for the full source of truth on all people and roles.

Name	Company	Role
Pablo	Deuna	CTO
Israel	Deuna	Data POC
Farhan	Deuna	Claude/LLM Access POC
Mark Walick	Deuna	PM Lead
Rakesh	Aidaptive	Engineer
Naoki	Aidaptive	Engineer

Milestones & Timeline

Milestone	Duration	Status	Notes
Phase 0: Assess level of effort/complexity	2 days	In Progress	$6K budget, started 2026-02-18
Phase 1: Model running in production for 2 processors with basic feature store	2 weeks	Pending	Core delivery
Phase 2: Add monitoring + integrate with experimentation	Week 3	Pending	Immediately after Phase 1
Phase 3: Drift detection, CI/CD, experiment ramp-up, additional model techniques	TBD	Pending	

Key Use Cases (P0)

ID	Use Case	Description
P-01	Outage detection & failover	Fail over/back via persistent timeout codes; random sampling of down PSP to detect recovery

ID	Use Case	Description
P-02	Overall transaction routing optimizer	Optimize Deuna's existing static rules based on historical outcomes
P-03	Per-transaction optimal route selection	Rank top 3 routes based on prior outcomes for fast retry
P-04	Message manipulation	Toggle CIT/MIT, AVS, MCC variables in authorization requests; top 3 recommendations
P-05	Retry optimization	Subs/MIT focused; enterprise darktime reduction; delayed retry based on reputation

---

Work Breakdown / Task Tracking

Backlog

- ☒ Claude access and LLM budget provisioned (2026-02-19)
- ☒ Confirm Snowflake read access provisioned — Rakesh verified (2026-02-18, info from Israel)
- ☒ Naoki Snowflake access confirmed (2026-02-19)
- ☐ Provision Deuna corp accounts for Rakesh and Naoki
  - ☐ Snowflake instance access for both accounts
  - ☒ Code (repo) access for Rakesh — granted by Pablo (2026-02-19)
  - ☐ Code (repo) access for Naoki
  - [~] Claude Code credits for both accounts — Not needed
- ☐ Complete Phase 0: assess level of effort/complexity (2 days, $6K)
- ☐ Build training platform (currently prototype-only — see Technical Gaps)
- ☐ Deliver P-01 through P-05 use cases

In Progress

- (nothing yet)

Done

- (nothing yet)

Schema Understanding & Data Notes

> Extracted 2026-02-18 from `PAYMENT_ML` Snowflake database. Full schema reference: SCHEMA.md

Overall Assessment

The schema is very well structured for the P0 use cases. The data is organized into clean source views in the `SOURCES` schema, and a massive denormalized flat table (`ABTESTING.ALL_VIEWS_FLAT`) that joins everything together — ideal for quick EDA and feature engineering without complex joins.

Key Tables for P0 Use Cases

`VW_ATHENA_PAYMENT_ATTEMPT` — most important table for routing & retry - Tracks every individual attempt with sequence order, processor used, error code/category, hard/soft decline type, retry indicator, and approved status - `DYNAMIC_ROUTING_DETAIL` (VARIANT/JSON) column likely contains rich routing decision metadata — needs exploration - `PAYMENT_ATTEMPT_SEQUENCE_ORDER` + `PAYMENT_LAST_ATTEMPT_INDICATOR` make it easy to reconstruct the full retry chain per payment - Directly supports **P-03** (per-transaction route selection) and **P-05** (retry optimization)

VW_SMART_ROUTING_ATTEMPTS — current routing engine event log - Captures per-attempt routing decisions: algorithm type, processor selected, process time, result status, skip reason - PROPERTIES_RESULT_PROCESS_TIME is a direct latency signal for the p95 $<$ 200ms target - PROPERTIES_RESULT_SKIPPED_REASON tells us why processors were bypassed — key for **P-01** (outage detection) - PROPERTIES_ALGORITHM_TYPE reveals what routing strategies are already in use

VW_ROUTING_MERCHANT_RULE + related views — existing rules engine - Deuna already has a rules-based routing system with conditions, members, options, and priority ordering - This is the foundation for **P-02** (optimize existing static rules) — we don't start from scratch - SHADOW_MODE in VW_ROUTING_MERCHANT_RULE_MEMBER suggests there's already infrastructure for testing new processors without live traffic

---

Feature Richness for ML Models

The data has strong signal across multiple dimensions:

Feature Group	Key Columns	Usefulness
Retry history	NUM_ATTEMPTS_ORDER, PREVIOUS_ORDER_ERROR_CODE, PREVIOUS_ORDER_PROCESSOR, AVG_SEC_BETWEEN_PAYMENT_ATTEMPS	Direct retry optimization signals
Error signals	ERROR_CODE, ERROR_CATEGORY, HARD_SOFT, EVENT_ERROR_STANDARD_ERROR_CODE	Distinguish hard vs soft declines; normalized error codes in events
Card signals	CARD_BIN, CARD_BRAND, BANK, CARD_COUNTRY	Processor affinity by card type
User behavior	TARGET_USER_FRAUD_RATE_COHORT, TARGET_USER_TENURE_IN_DAYS, TARGET_USER_FREQUENCY_VALUE, TOTA_MINUTES_BROWSING, TOTAL_NUM_SESSIONS	User risk and engagement signals
RFM	TARGET_USER_FREQUENCY_VALUE, TARGET_USER_RECENCY_VALUE, TARGET_USER_MONETARY_VALUE	Customer value for routing priority
Geo	LATITUDE, LONGITUDE, ORDER_COUNTRY_CODE, WEATHER_MAIN	Geography-based processor routing
Device	TARGET_USER_BROWSER, TARGET_USER_OS, TARGET_USER_DEVICE	Device fingerprinting
Message config	MCI_MSI_TYPE, ORDER_MCI_MSI_TYPE, PAYMENT_ATTEMPT_METHOD_TYPE	CIT/MIT toggle tracking for **P-04**
3DS	CHALLENGE_3DS_INDICATOR, CHALLENGE_3DS_STATUS	Available now; scoped to Phase 2

Starting Point Recommendation

- Use ABTESTING.ALL_VIEWS_FLAT for initial EDA — everything is already joined
- Switch to individual SOURCES views for model training to avoid data leakage and redundancy
- Explore DYNAMIC_ROUTING_DETAIL VARIANT column in VW_ATHENA_PAYMENT_ATTEMPT early — may contain routing features not exposed elsewhere

---

Notable Data Quality Observations

- **Typo in source data:** `PATMENT_TIME` in `VW_ATHENA_PAYMENT` (should be `PAYMENT_TIME`) — minor but worth noting for pipelines
- **Airline-specific data:** `VW_ORDER_AIRLINE_DETAIL_ALL` and `VW_ORDER_AIRLINE_INFORMATION_DETAIL_ALL` suggest Volaris is a key merchant with rich flight/passenger metadata
- `SOURCES` schema has no raw tables — only views, meaning underlying raw tables are managed upstream by Deuna's data team (Israel's domain)

Technical Gaps (from SOW)

Testing/Experimentation Platform — Production Ready

- A/B testing infrastructure, multi-variant experiments, automated winner selection
- Model registry with versioning, real-time inference (FastAPI sidecar)
- Missing: canary deployments

Training Platform — Prototype Only (needs significant work)

Current State

Component	Status	Notes
Basic training scripts	Prototype	3 models — NOT production-ready
Snowflake data access	Partial	Manual queries only
Logistic Regression model	Prototype	No real ML pipeline

Gap Tracking

#	Gap	Category	Priority	Status	Notes
G-01	Automated retraining	Automation	High	Not started	100% manual execution today
G-02	Orchestration	Infrastructure	High	Not started	No scheduling, retries, or workflow management
G-03	CI/CD pipeline	DevOps	High	Not started	No testing, no deployment automation
G-04	Data validation	Data Quality	High	Not started	No quality checks on inputs
G-05	Model monitoring	Observability	High	Not started	Can't detect model degradation
G-06	Deployment automation	Automation	High	Not started	Manual file uploads to EFS today
G-07	Model registration automation	Automation	Medium	Not started	Manual API calls today
G-08	Feature store	ML Infrastructure	High	Not started	Features recomputed each time — no caching
G-09	Drift detection	Observability	Medium	Not started	No alerts when data/model drifts

#	Gap	Category	Priority	Status	Notes
G-10	Lineage tracking	Governance	Medium	Not started	Can't trace which data produced which model
G-11	Hyperparameter tuning	ML Quality	Medium	Not started	Fixed parameters only
G-12	Algorithm comparison	ML Quality	Medium	Not started	Single algorithm (Logistic Regression) only
G-13	Versioning workflow	Governance	High	Not started	Manual version management
G-14	Rollback capability	Reliability	High	Not started	Can't revert bad models

Summary by Category

Category	Gaps	Notes
Automation	G-01, G-03, G-06, G-07	Core pipeline work — needed before any production use
Infrastructure	G-02, G-08	Orchestration + feature store are foundational
Observability	G-05, G-09	Monitoring + drift detection — needed post-deploy
Governance	G-10, G-13	Lineage + versioning — important for auditability
Reliability	G-14	Rollback — critical for safe production deployment
ML Quality	G-11, G-12	Nice to have in Phase 1; important for long-term quality
Data Quality	G-04	Validate inputs before training

Effort Assessment

To be filled in during Phase 0 assessment.

Gap	Estimated Effort	Dependencies	Owner
G-01 Automated retraining	TBD	G-02, G-08	TBD
G-02 Orchestration	TBD		TBD
G-03 CI/CD	TBD		TBD
G-04 Data validation	TBD	G-08	TBD
G-05 Model monitoring	TBD	G-06	TBD
G-06 Deployment automation	TBD	G-03	TBD
G-07 Model registration automation	TBD	G-06	TBD
G-08 Feature store	TBD		TBD
G-09 Drift detection	TBD	G-05	TBD
G-10 Lineage tracking	TBD	G-07, G-13	TBD
G-11 Hyperparameter tuning	TBD	G-01	TBD
G-12 Algorithm comparison	TBD	G-01	TBD
G-13 Versioning workflow	TBD	G-06	TBD
G-14 Rollback capability	TBD	G-13	TBD

Recommended Build Order

1. **Foundation:** G-02 Orchestration → G-08 Feature store → G-04 Data validation

2. **Automation:** G-03 CI/CD → G-06 Deployment automation → G-07 Model registration → G-01 Automated retraining
3. **Governance:** G-13 Versioning → G-10 Lineage → G-14 Rollback
4. **Observability:** G-05 Monitoring → G-09 Drift detection
5. **ML Quality:** G-11 Hyperparameter tuning → G-12 Algorithm comparison

---

Next Steps

Immediate Blockers to Resolve

- ☒ Claude/LLM access & budget — provisioned (2026-02-19)
- ☐ Provision Deuna corp accounts for Rakesh & Naoki (Snowflake, code/repo, Claude Code credits)
- ☒ Naoki Snowflake access confirmed (2026-02-19)

Phase 0 Assessment Work

- ☐ Explore `DYNAMIC_ROUTING_DETAIL` JSON column in `VW_ATHENA_PAYMENT_ATTEMPT` — likely contains rich routing metadata not exposed elsewhere
- ☐ Run EDA on `ABTESTING.ALL_VIEWS_FLAT` — understand data volumes, date ranges, merchant mix, approval rates, processor distribution
- ☐ Map existing routing rules — query `VW_ROUTING_MERCHANT_RULE` and related views to understand current rule engine
- ☐ Assess Athia training platform gaps — produce concrete list of what needs to be built vs. what already exists
- ☐ Size effort per use case (P-01 through P-05) — engineering, data, infra, and ML effort per workstream
- ☐ Identify risks and open questions — populate the Risks and Open Questions sections below

Documentation & Alignment

- ☐ Fill in Open Questions section — capture anything still unclear from the Deuna side
- ☐ Produce final Phase 0 deliverable — effort estimate doc with work breakdown, risks, and recommended build order for stakeholder sign-off

---

Decisions Log

Date	Decision	Rationale	Made By
2026-02-18	Latency target updated from p95 < 50ms to p95 < 200ms	Revised from original SOW spec	Rakesh (discussed with Pablo)
2026-02-19	Target merchant for Phase 1 set to **Volaris** (not Cinépolis)	Volaris has known PSPs (Worldpay ID:76, MIT ID:85, Elavon, Amex); Cinépolis only shows Cybersource gateway with unknown processor behind it	Mark Walick

---

Open Questions

#	Question	Owner	Status
1	Claude/LLM access & budget — when will this be provisioned?	Pablo → Farhan	**Done** (2026-02-19)

#	Question	Owner	Status
2	Are ATHIA_PREDICTIONS / ATHIA_FEEDBACK tables populated in Deuna's Snowflake today?	Israel / Rakesh	Open
3	Are SageMaker endpoints currently live for processor_selector / retry_predictor?	Rakesh	Open
4	Is there a live model in MODEL_ARTIFACTS that Deuna's payment service is calling today?	Rakesh	Open
5	What is the current payment volume through the routing engine? (Validates A/B test sample size feasibility)	Israel	Open
6	Who owns athena-platform Go repo deployments — Aidaptive or Deuna infra?	Pablo / Rakesh	Open

---

Risks & Issues

ID	Description	Likelihood	Impact	Mitigation	Status
R1	TBD	Low/Med/High	Low/Med/High	TBD	Open

Repository Analysis & Code Intelligence

> **Analyzed:** 2026-02-19 — Both Deuna repos cloned and analyzed in full. **Repos:** DATA-Athena-Snowflake | athena-platform

---

Repo 1: DATA-Athena-Snowflake — LLM Analytics Platform

What it is: A Python-based (FastAPI + LangGraph/LangChain) multi-agent AI platform. This is Athia's data intelligence layer — it uses LLMs (GPT-4o, Claude 3.7 Sonnet via Bedrock) to analyze Snowflake data, detect anomalies, and generate payment optimization strategies.

This is NOT a training platform. It produces LLM-generated strategies and insights — not trained ML models.

Architecture

- **Framework:** FastAPI + LangGraph (stimulus-response multi-agent pattern)
- **LLM Backend:** OpenAI GPT-4o (default), AWS Bedrock (Amazon Nova, Claude 3.7 Sonnet)
- **Data Layer:** Snowflake Snowpark with async session pooling, Jinja2-templated SQL queries
- **Deployment:** AWS Lambda + ECS + Bedrock AgentCore + SQS, CI/CD via GitHub Actions
- **Metrics Layer:** Centralized YAML-defined metrics with Jinja2 SQL templates (acceptance_rate, fraud_rate, 3ds_approval_rate, effective_cost_rate, chargeback_rate)

Implemented Workflows (11 stimuli)

Stimulus	Purpose	Status
`acceptance_rate_analysis_requested`	Detect acceptance rate drops, processor issues, BIN anomalies	Implemented (v0_1, v1_0)
`fraud_card_analysis_requested`	Fraud detection: card testing, false positives, geographic patterns	Implemented (v0_1)
`cost_optimization_analysis_requested`	Payment processing cost analysis	Early stage
`strategy_generation_initiated`	Orchestrate analysts and rank strategic recommendations	Partially implemented
`metrics_anomaly_research_triggered`	Automated anomaly detection and trend analysis	Implemented
`user_question_submitted`	Conversational chatbot for data queries	Implemented
`data_analyst_requested`	SQL generation and data visualization	Implemented
`researcher_assistance_requested`	Comprehensive research with parallel deep-dive explorers	Implemented
`deep_exploration_needed`	Root cause analysis for metric anomalies	Implemented
`element_edition_requested`	SQL query modification	Implemented
`knowledge_expert_asked`	External knowledge base via MCP	Implemented

Relevance to P0 Use Cases

- **P-02 (Routing optimizer):** `acceptance_rate_analysis_requested` workflow already analyzes processor performance and generates routing recommendations. This is a direct input to routing optimization.
- **P-05 (Retry optimization):** No retry-specific workflow exists yet. The `acceptance_rate_attempt` metric tracks retry attempts, but there is no dedicated retry analysis or routing workflow. **This is a gap we need to fill.**
- **P-01 (Outage detection):** No outage/failover workflow — this lives in the serving platform (athena-platform), not here.

Gaps Found in This Repo

- **Strategy Director incomplete:** The Matcher node has an `exit()` placeholder; Ranker uses dummy prompts.
- **No retry workflow:** No dedicated stimulus for retry optimization or retry routing.
- **No traditional ML:** Entirely LLM-based — no scikit-learn, XGBoost, neural nets. All pattern detection uses hardcoded thresholds (e.g., 15% drop threshold for acceptance rate).
- **No adaptive thresholds:** Fraud detection windows (60–80 min) and drop thresholds are hardcoded.
- **Limited error recovery:** No circuit breaker pattern; cascading failures bring down the whole workflow.

Repo 2: athena-platform — ML Serving & Experimentation Platform

What it is: A production Go (Gin) REST API that serves ML predictions, manages model experiments (A/B testing), and handles the full model lifecycle. This is the component that Deuna's payment service calls to get routing decisions in real time.

Architecture

- **Language/Framework:** Go + Gin (Clean Architecture)
- **Databases:** PostgreSQL (RDS Multi-AZ) + Redis (ElastiCache) for caching
- **ML Backends:** AWS SageMaker, Snowflake Cortex, custom HTTP endpoints
- **Snowflake Integration:** JWT-based private key auth; queries ATHIA_PREDICTIONS, ATHIA_FEEDBACK, ATHIA_TRAINING_DATASET, ATHIA_EXPERIMENT_LIFT
- **Deployment:** ECS Fargate + ALB (mTLS enforced for `/api/v1/ml/predict/*`) + EFS for model storage

ML Model Registry (Already Implemented)

Inference Type	Description	Maps to Use Case
`processor_selector`	Ranks available processors by approval probability	P-03: Per-transaction route selection
`retry_predictor`	Predicts retry success probability	P-05: Retry optimization
`retry_sequence`	Predicts optimal retry processor order	P-05: Retry optimization
`installment_optimizer`	Predicts best installment options	Out of scope Phase 1

Key finding: The model registry schema (`MODEL_ARTIFACTS`, `MODEL_EXPERIMENTS`, `MODEL_EXPERIMENT_VARIANTS`) and inference API are already built. **What's missing is the training pipeline** that produces models to register here.

A/B Experimentation System (Production-Ready)

- **Bucketing:** SHA256(transaction_id) % 10000 — deterministic and reproducible
- **Traffic splits:** Basis points (bps) — e.g., 30% control / 70% treatment
- **Auto-winner evaluation:** Statistical significance testing with full guardrails:
  - Minimum 7 days runtime
  - Minimum 1000 samples per variant
  - p-value < 0.05 (95% confidence)
  - Minimum 1% absolute lift
  - Latency regression guard: 10%
  - Revenue regression guard: -5%
- **Dry run mode:** Safe default for testing evaluation logic before enabling real rollouts
- **Merchant-specific experiments:** Experiments can be scoped to specific merchant IDs

Snowflake Tables for Training & Monitoring

Table	Purpose	Status
`ATHIA_PREDICTIONS`	Model inputs + outputs per prediction	Active
`ATHIA_FEEDBACK`	Transaction outcomes (approved/declined)	Active
`ATHIA_TRAINING_DATASET`	Predictions + feedback joined for retraining	Active
`ATHIA_EXPERIMENT_LIFT`	Aggregated experiment metrics for auto-winner	Active
`ATHIA_STAGE_OUTCOMES`	Per-stage funnel outcomes (installment → processor → retry)	**Designed, not deployed**
`ATHIA_SESSION_SUMMARY`	Full session-level aggregations	**Designed, not deployed**

Gaps Found in This Repo

- **Analytics tables not deployed:** `ATHIA_STAGE_OUTCOMES` and `ATHIA_SESSION_SUMMARY` tables exist in design docs but are not created in Snowflake. Multi-stage funnel analysis is blocked.
- **No production monitoring dashboards:** Grafana setup exists locally (docker-compose) but no alert rules or production dashboards are configured.
- **No training pipeline:** Model artifacts (SageMaker ARNs, Cortex endpoints) must currently be registered manually via API. No automated training → registration workflow.
- **Cache invalidation is manual:** Experiment assignment cache (Redis, 24h TTL) has no auto-invalidation on config changes.
- **LLM integration partial:** Bedrock client and prompt templates exist, but full agent orchestration is not complete.

- **No rate limiting:** No global rate limiting middleware — only per-user, per-feature quotas.

Impact on Effort Estimates

This analysis materially changes our understanding of the effort required. Here is what we learned:

What's Already Built (Reduces Effort)

Area	What Exists	Gap Gaps Affected
Model serving API	Full Go REST API with processor_selector + retry_predictor endpoints	Reduces P-03, P-05 integration work
Model registry	ModelArtifact + Experiment + ExperimentVariant tables + CRUD API	Reduces G-07 effort
A/B experimentation	Full auto-winner system with statistical guardrails	Reduces P-02 work
Snowflake feedback loop	ATHIA_PREDICTIONS + ATHIA_FEEDBACK + ATHIA_TRAINING_DATASET tables	Reduces G-05, G-08 baseline work
LLM analytics	Acceptance rate + fraud workflows for insights	Can accelerate P-02 analysis
CI/CD skeleton	GitHub Actions workflows for both repos	Reduces G-03 baseline work

What Still Needs to Be Built

Area	Specific Work	Gaps
Training pipeline	Automated end-to-end training → model registration	G-01, G-06, G-07
Feature store	Consistent feature computation + serving (Redis-backed)	G-08
Orchestration	Scheduled + trigger-based training runs	G-02
Analytics tables	Deploy ATHIA_STAGE_OUTCOMES + ATHIA_SESSION_SUMMARY in Snowflake	G-05
Production monitoring	Grafana dashboards + alerting rules for model performance	G-05, G-09
Retry workflow (LLM)	New stimulus `retry_optimization_requested` in DATA-Athena-Snowflake	P-05
Strategy Director	Finish matcher + ranker nodes in DATA-Athena-Snowflake	P-02
Data validation	Add schema + statistical validation to training pipeline	G-04
Versioning	Automated version bumping + metadata tagging on training runs	G-13
Lineage tracking	Record data → feature → model → deployment lineage	G-10
Rollback capability	One-command rollback to previous registered model	G-14
Drift detection	Feature + prediction distribution monitoring	G-09

Area	Specific Work	Gaps
Hyperparameter tuning	Integrate Optuna/Ray Tune into training pipeline	G-11
Algorithm comparison	Add XGBoost/LightGBM/NN alternatives to Logistic Regression	G-12

Testing Coverage & Architecture Deep-Dive

Analyzed: 2026-02-19 — Both repos examined in full for testing maturity and architectural patterns.

DATA-Athena-Snowflake — Testing Coverage

Metric	Detail
Test files	28
Test functions	421
Test frameworks	pytest 8.4, pytest-asyncio, pytest-mock
CI	Separate workflows for metrics + deployment only — no unified suite
Estimated coverage	~**18%**

Well tested: - Metrics layer: 70–80% (5 test files, strong validation tests) - Deployment config/validation: 50–60% - Deep-dive explorer utilities (pareto filter, metric router, data comparator)

Critical gaps (untested): - All 14 route handlers — 0% - All 13 services — 0% - All 3 clients (Redis, Postgres, Platform) — 0% - Core multi-agent framework (`AgentWorkflow`, `AgentStrategy`, `ToolStrategy`) — 1 manual test, not in pytest - All 11 workflow branches across versions — no unit tests - All 4 Lambda / AgentCore entrypoints — 0% - Middleware stack — 0%

Maturity: 2/5 — Immature. Only peripheral layers tested; the core product (multi-agent workflows) is essentially a black box.

DATA-Athena-Snowflake — Architecture

Pattern: Stimulus-Response multi-agent orchestration via LangGraph

```
Request → StimulusRegistry → OrchestratorWorkflow → Branch (DAG of Nodes)
        → AgentWorkflow (LangGraph StateGraph) → Response
```

11 registered stimuli (workflows), each with versioned implementations (e.g. `user_question_submitted` at v4.3/4.4/4.5). Each branch is a `BaseBranch` subclass composing `AgentStrategy` (LLM nodes) and `ToolStrategy` (deterministic tools) into a LangGraph DAG.

Key components: - `OrchestratorWorkflow` — dynamically builds workflow from YAML config + registry - `StimulusRegistry` — central map of stimulus → class, version, alias - `AgentWorkflow` — LangGraph `StateGraph` wrapper with `MemorySaver` checkpointing - `MessagesState` — carries full conversation history between nodes - FastAPI middleware stack: Auth → ValidateUsage → LogRequest → ApplyUsage → SessionLifecycle - LLM backends: OpenAI GPT-4o (default), AWS Bedrock (Claude 3.7 Sonnet, Amazon Nova Micro), Snowflake LLM

Notable architectural gaps: - Strategy Director matcher + ranker nodes have placeholder code — not functional (`exit()` in matcher, dummy prompts in ranker) - No retry-specific workflow — `retry_optimization_requested` stimulus is entirely missing (P-05 gap) - No traditional ML — all inference is LLM-based with hardcoded thresholds (e.g. 15% acceptance drop) - No circuit breaker — a cascading failure brings down the entire workflow - OpenTelemetry integration is commented out — no observability in production

---

athena-platform — Testing Coverage

Metric	Detail
Test files (`_test.go`)	126
Test functions	777
Test frameworks	Go `testing` + testify (assert/require/mock), in-memory SQLite for repos
CI	GitHub Actions on every PR — `make test/coverage`
Coverage threshold	**20%** (comment in code: "TODO: raise to 65")
`internal/clients/`	**Excluded from coverage entirely**

Well tested: - All 44 domain service interfaces have test files - ~43 PostgreSQL repository implementations tested via in-memory SQLite - V1 REST handlers: 15/18 (83%) - Auth middleware (JWT + API key) tested - Snowflake, Hermes, Merchant clients tested

Critical gaps (untested): - V2 API: 0/18 handlers — entire new API version has zero test coverage - **Bedrock client: 0%** — ML inference client excluded from coverage, no tests - 4 domain services untested: `auth`, `bedrock`, `element`, `workspace` - Bootstrap/DI integration test skipped (TODO: testcontainers) - 3 V1 handlers untested: `agent`, `workspaces`, `elements` - 0 benchmark tests — no performance regression safety net

Maturity: 3.5/5 — Solid foundation, with a critical blind spot in v2 + ML inference path.

---

athena-platform — Architecture

Pattern: Clean Architecture — strict layering

```
REST Handlers (v1 / v2, Gin)
        ↓
Controllers (~30 implementations)
        ↓
Domain Services (44 domain packages)
        ↓
Repositories (43 GORM implementations)
        ↓
PostgreSQL (RDS Multi-AZ) + Redis (ElastiCache)
```

Key design choices: - Constructor injection throughout — all services receive interface dependencies - 43 repository implementations, each domain entity has its own repo with query builders - Interface-driven — testify mocks and custom fake Redis for unit isolation - Event outbox table (`athia_event_outbox`) for reliable event delivery - Custom error types with HTTP status code mapping

Experiment / A-B testing system: - 4 model types: `processor_selector`, `retry_predictor`, `retry_sequence`, `installment_optimizer` - Experiment assignment: SHA256(transaction_id) % 10000 — deterministic bucketing - Redis-cached assignments with 24h TTL - Auto-winner worker (`cmd/worker/`) evaluates statistical significance → force-routes all traffic to winner - Rich stratification context: device, geo, card BIN, merchant tier, timing (hour/day/payday windows), customer LTV

Feedback loop: - Merchants POST feedback on predictions → PostgreSQL → feeds retraining signals via `SendFeedback()` / `SendBatchFeedback()`

Notable architectural gaps: - V2 API exists but fully untested — suggests incomplete refactoring - Bedrock client (ML path) untested and excluded from coverage config - No event-driven cache invalidation — stale experiment assignments possible for up to 24h after config changes - Tight coupling to `*gin.Context` — hard to test handlers without HTTP server - Experiment context passed as ad-hoc parameters — no middleware to propagate it automatically

---

Testing Summary — Both Repos

	DATA-Athena-Snowflake	athena-platform
Language	Python / FastAPI / LangGraph	Go / Gin
Test count	421 functions, 28 files	777 functions, 126 files
Estimated coverage	~18%	~25–30%
Core product tested?	No — multi-agent workflows untested	Partially — v2 API + Bedrock missing
Architecture pattern	Stimulus-response / LangGraph DAG	Clean Architecture / Repository
Biggest risk	Multi-agent core is a black box	V2 API + ML inference path untested
CI enforced?	Partial — fragmented workflows	Yes — every PR

Recommended immediate actions (both repos): 1. Add unit tests for core multi-agent framework nodes/edges (DATA-Athena-Snowflake) 2. Add tests for all 18 v2 handlers (athena-platform) 3. Add tests for Bedrock client + service (athena-platform) 4. Remove `internal/clients/` from coverage exclusions (athena-platform) 5. Raise coverage threshold from 20% → 60% with CI enforcement (athena-platform) 6. Build `retry_optimization_requested` stimulus — currently missing entirely (DATA-Athena-Snowflake)

Open Questions Raised by Repo Analysis

#	Question	Owner	Status
Q-R1	Are ATHIA_PREDICTIONS and ATHIA_FEEDBACK tables already populated in Deuna's Snowflake, or only in Athia's internal Snowflake?	Israel / Rakesh	Open
Q-R2	Are SageMaker endpoints currently live for processor_selector / retry_predictor, or are they placeholders?	Rakesh	Open
Q-R3	Is there a working model in MODEL_ARTIFACTS that Deuna's payment service is actually calling today?	Rakesh	Open
Q-R4	What is the current payment volume through the routing engine? (Needed to validate sample size requirements for A/B tests)	Israel	Open
Q-R5	Who owns the athena-platform Go repo deployment? Aidaptive or Deuna infra?	Pablo / Rakesh	Open

---

Notes & Meeting Log

2026-02-19 (late)

- **Portal live at https://deuna-ebce2.web.app** — Google Auth gate deployed. Only @aidaptive.com and @deuna.com email addresses can sign in. Built on Firebase Hosting + Firebase Auth JS SDK (signInWithPopup).
- Favicon added (D×A SVG, hosted at `/favicon.svg`).
- Scoping project GitHub repo link removed from portal References section (internal repo — not for client view).
- Note: `signInWithRedirect` was tried as an alternative to fix a COOP console warning, but it broke the sign-in flow and was reverted back to `signInWithPopup`. COOP warning is cosmetic and does not affect functionality.
- v12 PDF export generated and deployed to hosting portal.

2026-02-19 (afternoon)

- **Merchant selection: Volaris chosen as the target merchant for Phase 1** (over Cinépolis).
  - Cinépolis shows only Cybersource (a gateway — actual processor behind it is unknown), making it harder to work with.
  - Volaris has a clear, known set of PSPs:
    - **Worldpay** (Processor ID: 76)
    - **MIT** (Processor ID: 85)
    - **Elavon** — used for cards
    - **Amex** — used specifically for Amex cards
  - Volaris uses **4 processors total for cards**.
  - Routing policies exist for different currencies: MIT, Elavon, Worldpay handle different currency flows; Amex is dedicated to Amex card transactions.
  - This clarity makes Volaris the right starting point for P-03 (per-transaction route selection) and P-05 (retry optimization).

2026-02-19

- Analyzed both Deuna GitHub repositories in full: DATA-Athena-Snowflake and athena-platform.
- Key finding: athena-platform is a production-ready Go REST API with model registry, A/B testing, and auto-winner selection already built. The missing piece is the training pipeline.
- Key finding: DATA-Athena-Snowflake is an LLM-based analytics platform (not ML training). It generates strategies via GPT-4o / Claude, not trained models.
- The inference types `processor_selector` and `retry_predictor` already exist in the model registry schema — these map directly to P-03 and P-05.
- No retry-specific workflow exists in DATA-Athena-Snowflake — this needs to be built.
- Several analytics Snowflake tables (`ATHIA_STAGE_OUTCOMES`, `ATHIA_SESSION_SUMMARY`) are designed but not deployed. This affects multi-stage monitoring.
- Open questions added: live model status, ATHIA_ table status in Deuna Snowflake, payment volume for A/B test sizing.
- Pablo granted Rakesh code repository access (2026-02-19 morning).

Follow-up items to iterate on: - [ ] Are `ATHIA_PREDICTIONS` / `ATHIA_FEEDBACK` tables populated in Deuna's Snowflake today, or only in Athia's internal environment? (Ask Israel) - [ ] Are there live SageMaker endpoints behind `processor_selector` / `retry_predictor` today, or are they placeholders? (Rakesh to confirm) - [ ] What is the current payment volume through the routing engine? Minimum 1,000 transactions per variant needed for A/B test statistical validity. (Ask Israel) - [ ] Who owns athena-platform Go repo deployment — Aidaptive or Deuna infra? This affects Phase 1 deployment planning. (Clarify with Pablo) - [ ] Deploy `ATHIA_STAGE_OUTCOMES` and `ATHIA_SESSION_SUMMARY` tables in Snowflake — needed for multi-stage funnel monitoring (G-05). - [ ] Build `retry_optimization_requested` stimulus in DATA-Athena-Snowflake — no retry-specific LLM workflow exists today (P-05 gap). - [ ] Finish Strategy Director matcher + ranker nodes in DATA-Athena-Snowflake — currently has placeholder code (P-02 gap). - [ ] Confirm whether auto-winner worker is deployed in production with `DRY_RUN=false`, or still in dry-run mode.

2026-02-18

- Project plan file created. Details to be filled in.
- Israel is the main POC for data and related topics.
- Pablo is the CTO.
- All data is in Snowflake database; we will get read access to all tables. Snowflake URL: `VLTAXPW-RMONTES.snowflakecomputi`
- Need Claude access and budget for LLM. Farhan is the main POC; Pablo will be talking to Farhan to get this access.
- Mark Walick is the PM lead for this project.

Project Plan Exports

Date	File	Notes
2026-02-18	project-plan-2026-02-18.pdf	Initial export
2026-02-18	project-plan-2026-02-18-v2.pdf	Updated with schema notes, stakeholders, todos
2026-02-18	project-plan-2026-02-18-v3.pdf	Updated with TEAMS.md reference, Mark Walick correction
2026-02-18	project-plan-2026-02-18-v4.pdf	Self-contained: includes project plan + teams + schema
2026-02-18	project-plan-2026-02-18-v5.pdf	Updated project purpose to reflect scoping nature
2026-02-18	project-plan-2026-02-18-v6.pdf	Improved table formatting — fixed column overlaps
2026-02-18	project-plan-2026-02-18-v7.pdf	Latest snapshot
2026-02-18	project-plan-2026-02-18-v8.pdf	Refocused Phase 0 as assessment-only with clear deliverables
2026-02-18	project-plan-2026-02-18-v9.pdf	Added Next Steps section
2026-02-18	schema-2026-02-18.pdf	Initial Snowflake schema snapshot
2026-02-19	project-plan-2026-02-19-v12.pdf	Latest export
2026-02-19	project-plan-2026-02-19-v11.pdf	Access status updates, Volaris decision, follow-up items
2026-02-19	project-plan-2026-02-19-v10.pdf	Added full repo analysis: DATA-Athena-Snowflake + athena-platform findings, updated open questions

---

Documents & SOW Snapshots

Document	Date	Version	File
SOW: Athia Embedded into Acceptance - Phase 1	2026-02-16	v1	PDF

References & Links

- CLAUDE.md (project conventions)
- Deuna Code Repository (GitHub Org)

- Snowflake Data Repository
- Platform Repository
- Data Dictionary
- Athia Data Model
- Snowflake Login (`VLTAXPW-RMONTES.snowflakecomputing.com`)

Teams & Stakeholders

Source of truth for all people involved in the Smartrouter Scoping Project. **Last Updated:** 2026-02-18

Deuna

Name	Role	Responsibilities
Pablo	CTO	Executive sponsor; coordinating Claude/LLM access via Farhan
Israel	Data POC	Main point of contact for data and Snowflake access
Farhan	Claude/LLM Access POC	Provisioning Claude access and budget
Mark Walick	PM Lead	Product management lead

Aidaptive

Name	Role	Responsibilities
Rakesh	Engineer	Engineering lead; Snowflake access verified
Naoki	Engineer	Engineering; Snowflake access pending test

Key Contacts by Topic

Topic	Owner	Notes
Data / Snowflake	Israel (Deuna)	All data questions, schema, access
Claude / LLM Budget	Farhan (Deuna)	Pablo coordinating with Farhan
Project Management	Mark Walick (Deuna)	
Engineering	Rakesh + Naoki (Aidaptive)	Coordinate with each other on access/setup
Executive Decisions	Pablo (Deuna)	CTO sign-off

Snowflake Schema Reference

Database: `PAYMENT_ML` **Instance:** `VLTAXPW-RMONTES.snowflakecomputing.com` **Extracted:** 2026-02-18

Overview

Schema	Type	Object	Columns
ABTESTING	Table	ALL_VIEWS_FLAT	~319 (denormalized flat table)
ABTESTING	Table	ALL_VIEWS_FLAT_SAMPLE	~319 (sample of above)
SOURCES	View	VW_ATHENA_CHANNEL	2
SOURCES	View	VW_ATHENA_ORDER	85
SOURCES	View	VW_ATHENA_ORDER_COMPLEMENT	11
SOURCES	View	VW_ATHENA_PAYMENT	46
SOURCES	View	VW_ATHENA_PAYMENT_ATTEMPT	39
SOURCES	View	VW_ATHENA_PAYMENT_EVENTS	28
SOURCES	View	VW_ATHENA_TARGET_USER	40
SOURCES	View	VW_ORDER_AIRLINE_DETAIL_ALL	29
SOURCES	View	VW_ORDER_AIRLINE_INFORMATION_DETAIL_ALL	51
SOURCES	View	VW_ROUTING_MERCHANT_RULE	14
SOURCES	View	VW_ROUTING_MERCHANT_RULE_CONDITION	16
SOURCES	View	VW_ROUTING_MERCHANT_RULE_MEMBER	15
SOURCES	View	VW_ROUTING_MERCHANT_RULE_OPTION	8
SOURCES	View	VW_ROUTING_MERCHANT_RULE_OPTION_VALUES	8
SOURCES	View	VW_SMART_ROUTING_ATTEMPTS	40

Schema: ABTESTING

Denormalized flat tables joining all Athena views — used for A/B testing analysis.

ALL_VIEWS_FLAT / ALL_VIEWS_FLAT_SAMPLE

Both tables share the same ~319 columns. `ALL_VIEWS_FLAT_SAMPLE` is a sampled subset.

Key column groups:

Group	Columns
Identity	`SOURCE_TABLE_NAME`, `CHANNEL_ID`, `CHANNEL_NAME`, `COMMERCE_ID`, `TARGET_USER_ID`, `USER_ACCOUNT_ID`
Order	`ORDER_ID`, `ORDER_DATE`, `ORDER_TIME`, `ORDER_STATUS`, `ORDER_TOKEN`, `COMMERCE_STORE_CODE`
Order Indicators	`ORDER_APPROVED_INDICATOR`, `ORDER_REJECTED_INDICATOR`, `ORDER_SEND_TO_SMART_ROUTING_INDICATOR`, `ORDER_RECOVERED_BY_SMART_ROUTING_INDICATOR`, `ORDER_APPROVED_BY_FIRST_PROCESSOR_INDICATOR`, `ORDER_DENIED_BY_FRAUD_INDICATOR`, `ORDER_DENIED_BY_PROCESSOR_INDICATOR`
Order Amounts	`ORDER_ORIGINAL_GMV_AMOUNT`, `ORDER_GMV_AMOUNT_USD`, `ORDER_AUTH_AMOUNT_USD`, `ORDER_CAPTURE_AMOUNT_USD`, `ORDER_TOTAL_AMOUNT_USD`
Payment	`PAYMENT_ID`, `PAYMENT_DATE`, `PAYMENT_STATUS`, `PROCESSOR_NAME`, `PAYMENT_AMOUNT_USD`

Group	Columns
Payment Attempt	PAYMENT_ATTEMPT_ID, PAYMENT_ATTEMPT_SEQUENCE_ORDER, PAYMENT_ATTEMPT_STATUS, PAYMENT_ATTEMPT_PROCESSOR_NAME, PAYMENT_ATTEMPT_ERROR_CODE, PAYMENT_ATTEMPT_APPROVED_INDICATOR
Event	EVENT_TYPE, EVENT_STATUS, EVENT_CREATED_AT, EVENT_ERROR_CODE, EVENT_ERROR_STANDARD_ERROR_CODE
Card	CARD_BIN, CARD_BRAND, CARD_LAST_FOUR, CARD_COUNTRY, BANK
Fraud	FRAUD_PROCESSOR_NAME, FRAUD_RISK_LEVEL, FRAUD_RISK_SCORE, FRAUD_STATUS
User	TARGET_USER_BROWSER, TARGET_USER_OS, TARGET_USER_DEVICE, TARGET_USER_FRAUD_RATE_COHORT, TARGET_USER_TENURE_IN_DAYS
Routing Rules	RULE_ID, PROPERTIES__RULES_LABEL, MERCHANT_PAYMENT_PROCESSOR_NAME, COMMERCE_ROUTING_MERCHANT_RULE_VERSION_ID
Geo	LATITUDE, LONGITUDE, ORDER_CITY_NAME, ORDER_STATE_NAME, ORDER_COUNTRY_CODE, WEATHER_MAIN
Airline	PNR, FLIGHT_NUMBER, CARRIER_CODE, DESTINATION_IATA_CODE, TOTAL_PASSENGER

Schema: SOURCES

Raw source views feeding the ABTESTING schema. Join key across most views: COMMERCE_ID, ORDER_ID, PAYMENT_ID, PAYMENT_ATTEMPT_ID.

VW_ATHENA_CHANNEL (2 cols)

Channel lookup table.

Column	Type
CHANNEL_ID	NUMBER(5,0)
CHANNEL_NAME	VARCHAR

VW_ATHENA_ORDER (85 cols)

Core order-level data including status, amounts, payment method, behavioral signals, and geo.

Column	Type	Notes
COMMERCE_ID	VARCHAR	Merchant ID
TARGET_USER_ID	VARCHAR(32)	User ID
USER_ACCOUNT_ID	VARCHAR(32)	
CHANNEL_ID	NUMBER	
ORDER_ID	VARCHAR	Primary key
ORDER_DATE / ORDER_TIME	DATE / TIME	
ORDER_STATUS	VARCHAR	

Column	Type	Notes
ORDER_APPROVED_INDICATOR	BOOLEAN	
ORDER_SEND_TO_SMART_ROUTING_INDICATOR	BOOLEAN	Was smart routing used?
ORDER_RECOVERED_BY_SMART_ROUTING_INDICATOR	BOOLEAN	Did smart routing recover?
ORDER_DENIED_BY_FRAUD_INDICATOR	BOOLEAN	
ORDER_ORIGINAL_GMV_AMOUNT / _USD	FLOAT	
ORDER_AUTH_AMOUNT_USD	FLOAT	
ORDER_TOTAL_AMOUNT_USD	FLOAT	
PAYMENT_CURRENCY	VARCHAR	
CARD_LAST_FOUR / CARD_COUNTRY	VARCHAR	
DEVICEID / REQUEST_IP	VARCHAR	
USER_IS_GUEST	BOOLEAN	
TOTA_MINUTES_BROWSING	NUMBER	Behavioral feature
TOTAL_EVENTS_BEFORE_PURCHASE	NUMBER	Behavioral feature
TOTAL_NUM_SESSIONS	NUMBER	Behavioral feature
LATITUDE / LONGITUDE	NUMBER	
WEATHER_MAIN	VARCHAR	
ORDER_TOKEN	VARCHAR(100)	

VW_ATHENA_ORDER_COMPLEMENT (11 cols)

Fraud and 3DS signals at the order level.

Column	Type
COMMERCE_ID	VARCHAR
CHANNEL_ID	NUMBER
ORDER_ID	VARCHAR
FRAUD_PROCESSOR_NAME	VARCHAR
FRAUD_RISK_LEVEL	VARCHAR
FRAUD_RISK_SCORE	FLOAT
FRAUD_STATUS	VARCHAR
SITEDOMAIN	VARCHAR
WEBSITENAME	VARCHAR
CHALLENGE_3DS_INDICATOR	BOOLEAN
CHALLENGE_3DS_STATUS	VARCHAR

VW_ATHENA_PAYMENT (46 cols)

Payment-level data: processor, card info, error codes, routing rules.

Column	Type	Notes
PAYMENT_ID	VARCHAR(250)	Primary key
ORDER_ID	VARCHAR	FK → Order
PAYMENT_DATE / PATMENT_TIME	DATE / TIME	Note: typo in source (`PATMENT`)

Column	Type	Notes
PAYMENT_STATUS	VARCHAR	
PROCESSOR_NAME	VARCHAR	
CARD_BIN / CARD_BRAND / BANK	VARCHAR	
NUM_ATTEMPTS_ORDER	NUMBER	
NUM_ATTEMPTS_SMART_ROUTING	NUMBER	
ERROR_MESSAGE / ERROR_CODE / ERROR_CATEGORY	VARCHAR	
PAYMENT_AMOUNT_USD	FLOAT	
HARD_SOFT	VARCHAR	Hard vs soft decline
RULE_ID	VARCHAR	Routing rule applied
PROPERTIES___RULES_LABEL	VARCHAR	
MERCHANT_PAYMENT_PROCESSOR_NAME	VARCHAR	
MERCHANT_PAYMENT_PROCESSOR_ID	VARCHAR	
PREVIOUS_ORDER_ERROR_CODE	VARCHAR	Prior attempt context
PREVIOUS_ORDER_PROCESSOR	VARCHAR	
AUTHORIZATION_CODE	VARCHAR	
COMMERCE_ROUTING_MERCHANT_RULE_VERSION_ID	VARCHAR(36)	

VW_ATHENA_PAYMENT_ATTEMPT (39 cols)

Individual attempt-level data — key table for retry optimization.

Column	Type	Notes
PAYMENT_ATTEMPT_ID	VARCHAR(32)	Primary key
PAYMENT_ID	VARCHAR(250)	FK → Payment
ORDER_ID	VARCHAR	FK → Order
PAYMENT_ATTEMPT_SEQUENCE_ORDER	NUMBER	Attempt number
PAYMENT_LAST_ATTEMPT_INDICATOR	BOOLEAN	
PAYMENT_ATTEMPT_STATUS	VARCHAR	
PAYMENT_ATTEMPT_PROCESSOR_NAME	VARCHAR	Which processor used
PAYMENT_ATTEMPT_PROCESSOR_CODE	VARCHAR	
PAYMENT_ATTEMPT_ERROR_CODE	VARCHAR	
PAYMENT_ATTEMPT_ERROR_CATEGORY	VARCHAR	
PAYMENT_ATTEMPT_HARD_SOFT_TYPE	VARCHAR	
PAYMENT_ATTEMPT_RETRY_INDICATOR	VARCHAR	
PAYMENT_ATTEMPT_APPROVED_INDICATOR	BOOLEAN	
PAYMENT_ATTEMPT_ACCEPTANCE_RATE_INDICATOR	BOOLEAN	
PAYMENT_ATTEMPT_AMOUNT_USD	FLOAT	
PAYMENT_ATTEMPT_CARD_BRAND / CARD_BIN / BANK	VARCHAR	
DENIED_BY_PSP_OR_FRAUD	VARCHAR	
DYNAMIC_ROUTING_DETAIL	VARIANT	JSON routing detail
RULE_ID	VARCHAR	
MERCHANT_PAYMENT_PROCESSOR_ID	VARCHAR	
COMMERCE_ROUTING_MERCHANT_RULE_VERSION_ID	VARCHAR(36)	

VW_ATHENA_PAYMENT_EVENTS (28 cols)

Event stream for each payment attempt — captures state transitions.

Column	Type	Notes
PAYMENT_ATTEMPT_ID	VARCHAR(32)	FK → Attempt
PAYMENT_ATTEMP_EVENT_INDEX	NUMBER	Event order within attempt
EVENT_TYPE	VARCHAR	
EVENT_STATUS	VARCHAR	
EVENT_CREATED_AT	TIMESTAMP_NTZ	
EVENT_ORIGINAL_TOTAL_AMOUNT	NUMBER	
EVENT_ERROR_CODE	VARCHAR	
EVENT_ERROR_STANDARD_ERROR_CODE	VARCHAR	Normalized error code
EVENT_ERROR_STANDARD_ERROR_MESSAGE	VARCHAR	
EVENT_ERROR_DEUNA	VARCHAR	Deuna-specific error
EVENT_REFUND_VOID_REASON	VARCHAR	

VW_ATHENA_TARGET_USER (40 cols)

User profile and behavioral signals.

Column	Type	Notes
TARGET_USER_ID	VARCHAR(32)	Primary key
COMMERCE_ID	VARCHAR	
TARGET_USER_BROWSER / OS / DEVICE / EQUIPMENT	VARCHAR	Device fingerprint
TARGET_USER_FAVORITE_PAYMENT_METHOD	VARCHAR	
TARGET_USER_FAVORITE_CARD_BRAND / BANK	VARCHAR	
TARGET_USER_ACCESS_COUNTRY_CODE	VARCHAR	
TARGET_USER_FIRST_PURCHASE_DATE	TIMESTAMP	
TARGET_USER_LAST_PURCHASE_DATE	TIMESTAMP	
TARGET_USER_USER_FRAUD_RATE	NUMBER	
TARGET_USER_FRAUD_RATE_COHORT	VARCHAR(30)	
TARGET_USER_TENURE_IN_DAYS	NUMBER	
TARGET_USER_FREQUENCY_VALUE	NUMBER	RFM frequency
TARGET_USER_RECENCY_VALUE	NUMBER	RFM recency
TARGET_USER_MONETARY_VALUE	FLOAT	RFM monetary
TARGET_USER_NUM_ORDERS_VALUE	NUMBER	

VW_ORDER_AIRLINE_DETAIL_ALL (29 cols)

Airline booking details (Volaris-specific). Joined via `ORDER_ID`.

Key fields: `PNR`, `BOOKINGISINTERNATIONAL`, `NAVITAIRE_CARRIER_CODE`, `TOTAL_FLIGHT_NUMBERS`, `TOTAL_PASSENGER`, `ROUND_FLIGHT_IND`

VW_ORDER_AIRLINE_INFORMATION_DETAIL_ALL (51 cols)

Flight + passenger details per order. Joined via `ORDER_ID`.

Key fields: `FLIGHT_NUMBER`, `CARRIER_CODE`, `ORIGIN_IATA_CODE`, `DESTINATION_IATA_CODE`, `PASSENGER_TYPE`, `PASSENGER_FREQUENT_FLYER_CODE`, `SERVICE_CLASS`, `TOTAL_AMOUNT_USD`

VW_ROUTING_MERCHANT_RULE (14 cols)

Merchant routing rules configuration.

Column	Type
ID	NUMBER
MERCHANT_ID	VARCHAR
LABEL	VARCHAR
STATUS	VARCHAR
PRIORITY	NUMBER
TRIGGER_	VARCHAR
IS_DEFAULT	VARCHAR
IGNORE_NEXT_RULES	VARCHAR
MERCHANT_RULE_PARENT	NUMBER
CREATED_AT / UPDATED_AT / DELETED_AT	TIMESTAMP

VW_ROUTING_MERCHANT_RULE_CONDITION (16 cols)

Conditions that trigger routing rules.

Key fields: `MERCHANT_RULE_ID`, `MERCHANT_RULE_OPTION_ID`, `OPERAND`, `OPERAND_FIELD_TO_EVALUATE`, `OPERATOR`, `METADATA_FIELD_NAME`

VW_ROUTING_MERCHANT_RULE_MEMBER (15 cols)

Processors assigned to routing rules.

Key fields: `MERCHANT_RULE_ID`, `PAYMENT_PROCESSOR_ID`, `MERCHANT_PAYMENT_PROCESSOR_ID`, `STRATEGY`, `SORT`, `SHADOW_MODE`, `CAPABILITIES`, `FRAUD_PROCESSOR`

VW_ROUTING_MERCHANT_RULE_OPTION (8 cols)

Available routing rule option types.

Key fields: `ID`, `LABEL`, `OPERATORS_AVAILABLE`

VW_ROUTING_MERCHANT_RULE_OPTION_VALUES (8 cols)

Allowed values for routing rule options.

Key fields: `ID`, `MERCHANT_RULE_OPTION`, `VALUE_`

VW_SMART_ROUTING_ATTEMPTS (40 cols)

Event stream from the smart routing engine — per-attempt routing decisions.

Column	Type	Notes
ATTEMPT_ID	NUMBER	
PROPERTIES_TRANSACTION_ID	VARCHAR	Links to payment
PROPERTIES_MERCHANT_ID	VARCHAR	
PROPERTIES_ALGORITHM_TYPE	VARCHAR	Which routing algorithm
RULE_ID	NUMBER	Rule applied
PROPERTIES_GATEWAY	BOOLEAN	
PROPERTIES_PAYMENT_PROCESSOR_ID	NUMBER	
PROPERTIES_PROCESSOR_CODE	VARCHAR	
PROPERTIES_RESULT_STATUS	VARCHAR	
PROPERTIES_RESULT_ERROR_CODE	VARCHAR	
PROPERTIES_RESULT_PROCESS_TIME	FLOAT	Latency signal
PROPERTIES_RESULT_SKIPPED_REASON	VARCHAR	Why processor was skipped
PROPERTIES_FRANCHISE / COUNTRY / CITY / STATE	VARCHAR	
PROPERTIES_ORDER_VALUE	NUMBER	
ORIGINAL_TIMESTAMP / RECEIVED_AT	TIMESTAMP	

Key Relationships

```
VW_ATHENA_CHANNEL
      CHANNEL_ID → VW_ATHENA_ORDER

VW_ATHENA_ORDER
      ORDER_ID → VW_ATHENA_ORDER_COMPLEMENT
      ORDER_ID → VW_ATHENA_PAYMENT
      ORDER_ID → VW_ORDER_AIRLINE_DETAIL_ALL
      ORDER_ID → VW_ORDER_AIRLINE_INFORMATION_DETAIL_ALL
      TARGET_USER_ID → VW_ATHENA_TARGET_USER

VW_ATHENA_PAYMENT
      PAYMENT_ID → VW_ATHENA_PAYMENT_ATTEMPT
      RULE_ID → VW_ROUTING_MERCHANT_RULE

VW_ATHENA_PAYMENT_ATTEMPT
      PAYMENT_ATTEMPT_ID → VW_ATHENA_PAYMENT_EVENTS
      PROPERTIES_TRANSACTION_ID → VW_SMART_ROUTING_ATTEMPTS

VW_ROUTING_MERCHANT_RULE
      ID → VW_ROUTING_MERCHANT_RULE_CONDITION
      ID → VW_ROUTING_MERCHANT_RULE_MEMBER

ABTESTING.ALL_VIEWS_FLAT
      Denormalized join of all above views
```